# ASSIGNMENT- 9.1

**Name : Ch.Akhila**

**H.T.NO: 2303A51234**

**Batch : 18**

**Problem 1:**

Consider the following Python function:

def find_max(numbers):

return max(numbers)

Task:

• Write documentation for the function in all three formats:

(a) Docstring

(b) Inline comments

(c) Google-style documentation

• Critically compare the three approaches. Discuss the

advantages, disadvantages, and suitable use cases of each

style.

• Recommend which documentation style is most effective

for a mathematical utilities library and justify your

answer

## Promt :

# Write documentation for the function in three formats

# and compare them. Also recommend the best style.


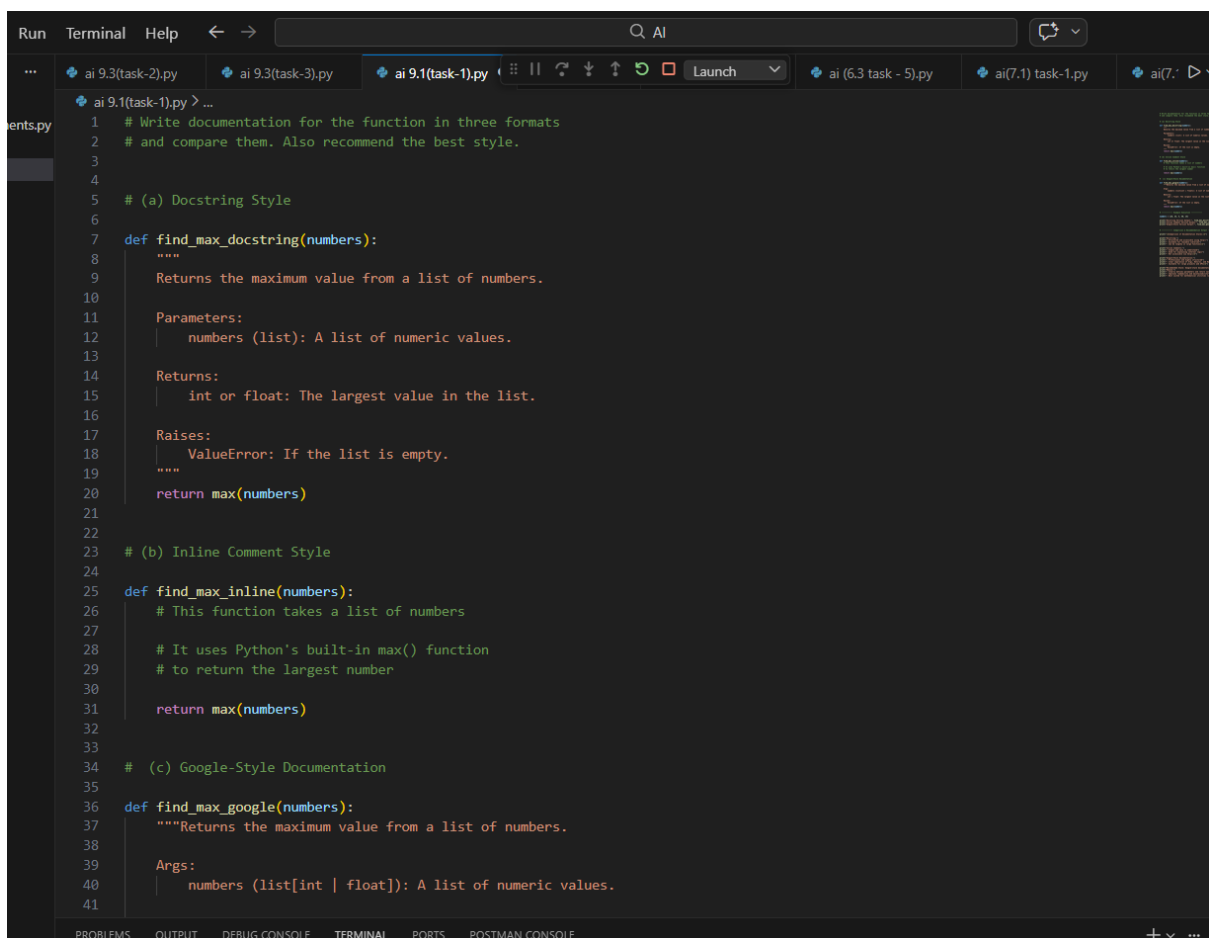# (a) Docstring Style

# (b) Inline Comment Style

# This function takes a list of numbers

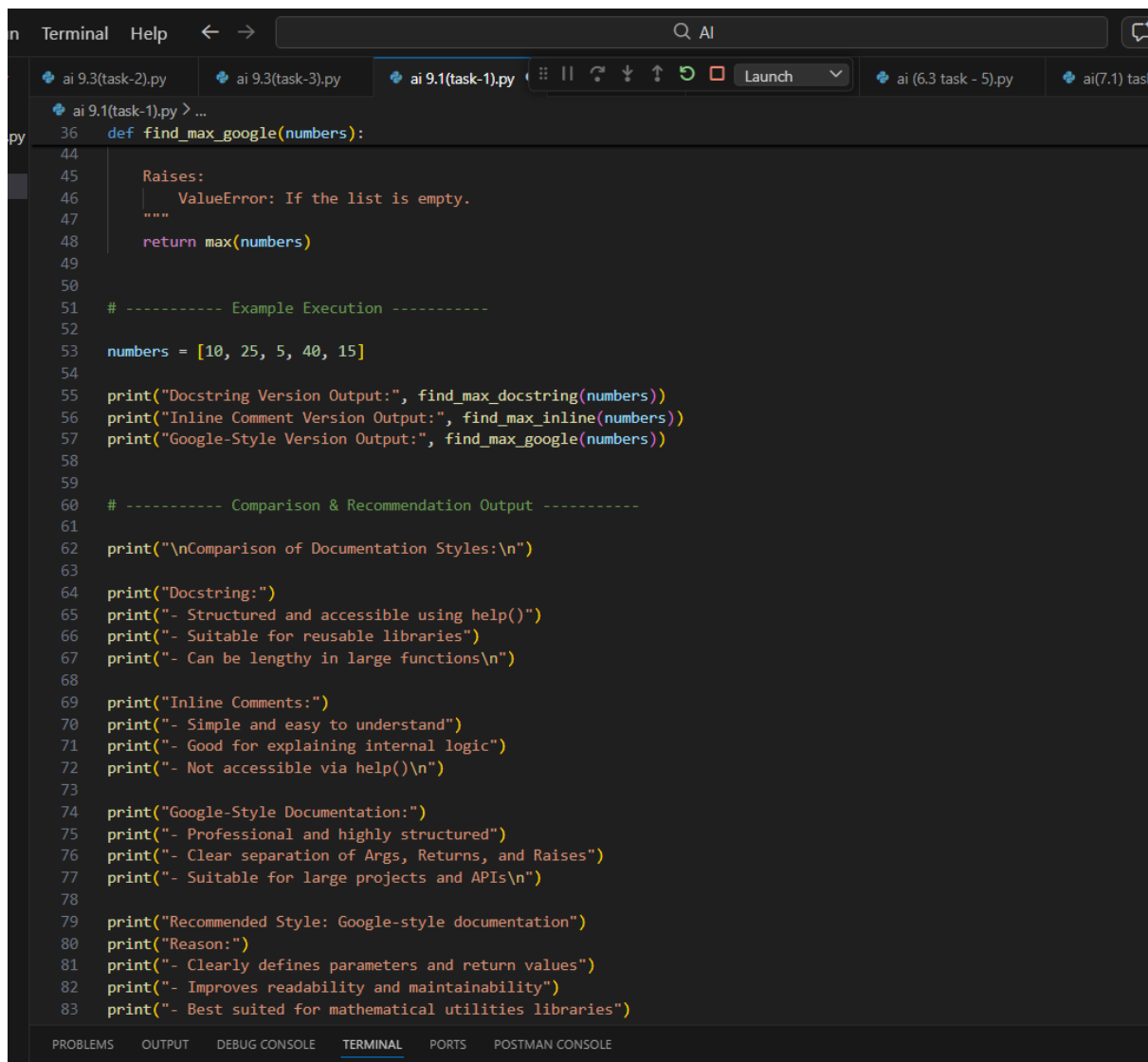   # It uses Python's built-in max() function

   # to return the largest number
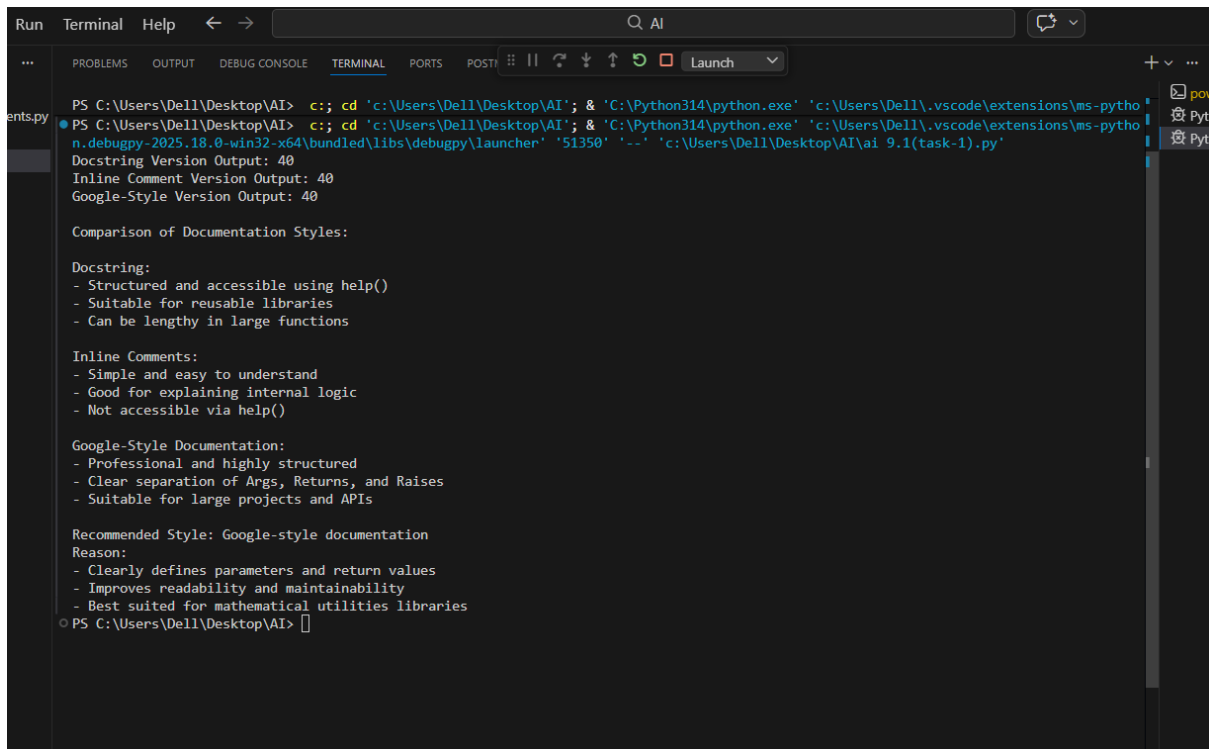
#  (c) Google-Style Documentation

# INPUT:

```python
# Write documentation for the function in three formats
# and compare them. Also recommend the best style.


# (a) Docstring Style

def find_max_docstring(numbers):
    """
    Returns the maximum value from a list of numbers.

    Parameters:
        numbers (list): A list of numeric values.

    Returns:
        int or float: The largest value in the list.

    Raises:
        ValueError: If the list is empty.
    """
    return max(numbers)


# (b) Inline Comment Style

def find_max_inline(numbers):
    # This function takes a list of numbers

    # It uses Python's built-in max() function
    # to return the largest number

    return max(numbers)


#  (c) Google-Style Documentation

def find_max_google(numbers):
    """Returns the maximum value from a list of numbers.

    Args:
        numbers (list[int | float]): A list of numeric values.
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   **TERMINAL**   PORTS   POSTMAN CONSOLE

```
  ai 9.3(task-2).py       ai 9.3(task-3).py       ai 9.1(task-1).py    ⠿ II ⌃ ↓ ↑ ↻ ☐   Launch  ⌄       ai (6.3 task - 5).py       ai(7.1) task

   ai 9.1(task-1).py > ...
36     def find_max_google(numbers):
44
45         Raises:
46             ValueError: If the list is empty.
47         """
48         return max(numbers)
49
50
51     # ----------- Example Execution -----------
52
53     numbers = [10, 25, 5, 40, 15]
54
55     print("Docstring Version Output:", find_max_docstring(numbers))
56     print("Inline Comment Version Output:", find_max_inline(numbers))
57     print("Google-Style Version Output:", find_max_google(numbers))
58
59
60     # ----------- Comparison & Recommendation Output -----------
61
62     print("\nComparison of Documentation Styles:\n")
63
64     print("Docstring:")
65     print("- Structured and accessible using help()")
66     print("- Suitable for reusable libraries")
67     print("- Can be lengthy in large functions\n")
68
69     print("Inline Comments:")
70     print("- Simple and easy to understand")
71     print("- Good for explaining internal logic")
72     print("- Not accessible via help()\n")
73
74     print("Google-Style Documentation:")
75     print("- Professional and highly structured")
76     print("- Clear separation of Args, Returns, and Raises")
77     print("- Suitable for large projects and APIs\n")
78
79     print("Recommended Style: Google-style documentation")
80     print("Reason:")
81     print("- Clearly defines parameters and return values")
82     print("- Improves readability and maintainability")
83     print("- Best suited for mathematical utilities libraries")
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    **TERMINAL**    PORTS    POSTMAN CONSOLE

**OUTPUT:**

```
PS C:\Users\Dell\Desktop\AI>  c:; cd 'c:\Users\Dell\Desktop\AI'; & 'C:\Python314\python.exe' 'c:\Users\Dell\.vscode\extensions\ms-pytho
● PS C:\Users\Dell\Desktop\AI>  c:; cd 'c:\Users\Dell\Desktop\AI'; & 'C:\Python314\python.exe' 'c:\Users\Dell\.vscode\extensions\ms-pytho
  n.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51350' '--' 'c:\Users\Dell\Desktop\AI\ai 9.1(task-1).py'
  Docstring Version Output: 40
  Inline Comment Version Output: 40
  Google-Style Version Output: 40

  Comparison of Documentation Styles:

  Docstring:
  - Structured and accessible using help()
  - Suitable for reusable libraries
  - Can be lengthy in large functions

  Inline Comments:
  - Simple and easy to understand
  - Good for explaining internal logic
  - Not accessible via help()

  Google-Style Documentation:
  - Professional and highly structured
  - Clear separation of Args, Returns, and Raises
  - Suitable for large projects and APIs

  Recommended Style: Google-style documentation
  Reason:
  - Clearly defines parameters and return values
  - Improves readability and maintainability
  - Best suited for mathematical utilities libraries
○ PS C:\Users\Dell\Desktop\AI> []
```

## EXPLANATION:

### Docstring Style – Explanation

Docstrings are written inside triple quotes below a function and provide a clear description of what the function does. They explain inputs, outputs, and possible errors in a readable format. Docstrings can be accessed using help() and are useful for documentation tools. They are good for reusable functions and structured code.

### Inline Comment Style – Explanation

Inline comments are written using # inside the function body. They explain the logic of specific lines of code and improve readability. However, they are not structured documentation and cannot be accessed automatically. They are mainly useful for explaining complex steps in the code.

### Google-Style Documentation – Explanation

Google-style documentation is a well-structured docstring format that clearly separates sections like Args, Returns, and Raises. It improves clarity and consistency in larger projects. This style is widely used in professional development. It is best suited for libraries and production-level code

## Problem 2:

Consider the following Python function:

```
def login(user, password, credentials):

return credentials.get(user) == password
```

Task:

1. Write documentation in all three formats.

2. Critically compare the approaches.

3. Recommend which style would be most helpful for new

developers onboarding a project, and justify your choice.

**PROMT:**

```
# Write documentation for the login function using standard Python docstring format.

#Rewrite the same login function documentation using inline comments.

#Rewrite the login function using Google-style documentation format.

#Create example credentials data and test all three versions of the function.

#Compare docstring, inline comments, and Google-style documentation

# in terms of structure, clarity, and maintainability.

# Discuss advantages and disadvantages of each documentation style.

#Recommend the most suitable documentation style

# for onboarding new developers to a project.


# Justify the recommendation with clear reasoning.
```
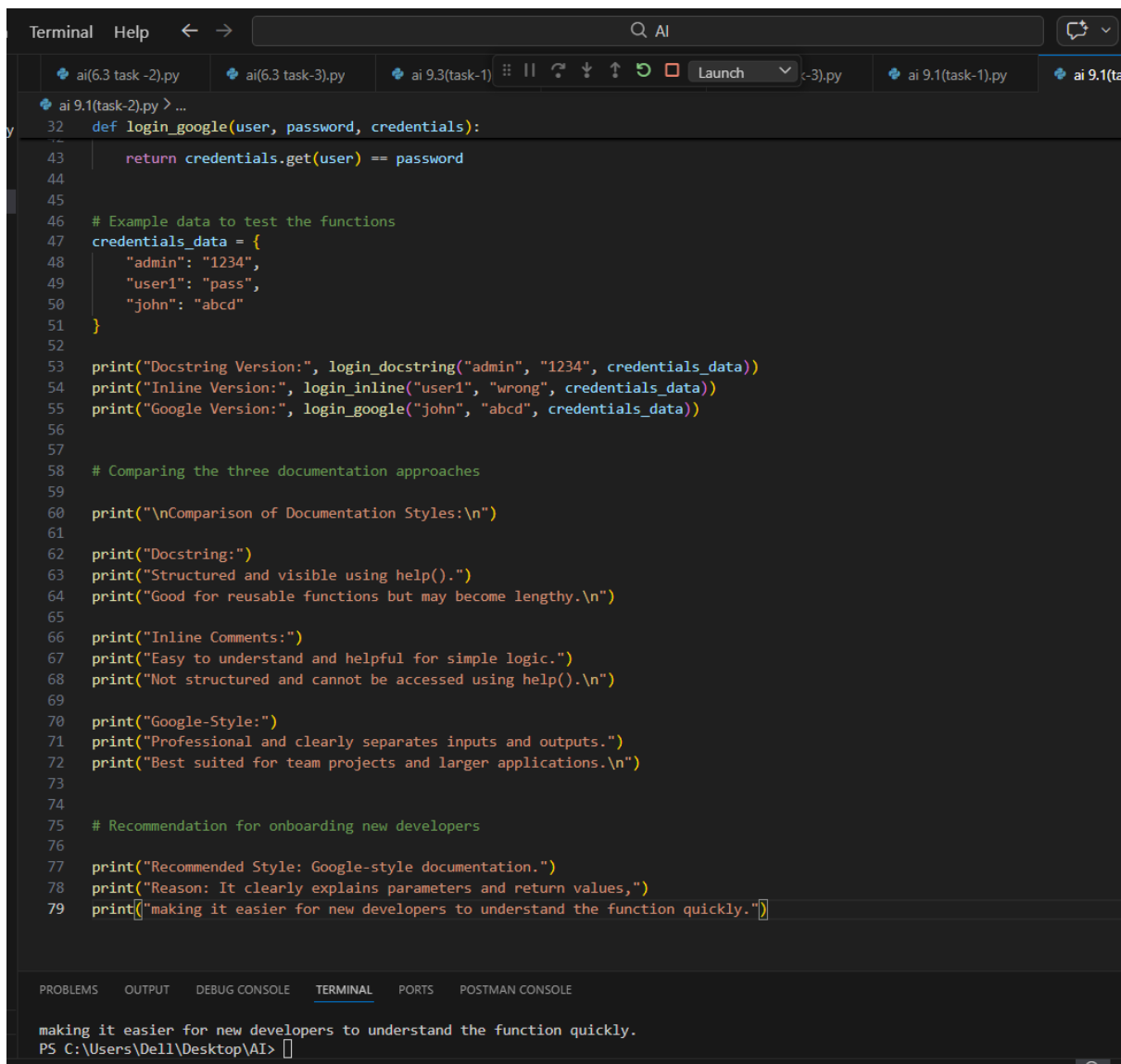
**INPUT:**

◆ ai 9.1(task-2).py > ...

```python
 1   # Write documentation for this login function in different formats
 2   # and compare them. Finally, suggest which style is best for
 3   # new developers joining a project.
 4
 5
 6   # Docstring style documentation
 7   def login_docstring(user, password, credentials):
 8       """
 9       This function checks whether the given username and password
10       match the stored credentials.
11
12       Parameters:
13           user (str): Username of the person logging in.
14           password (str): Password entered by the user.
15           credentials (dict): Dictionary containing username-password pairs.
16
17       Returns:
18           bool: True if login is successful, otherwise False.
19       """
20       return credentials.get(user) == password
21
22
23   # Same function explained using inline comments
24   def login_inline(user, password, credentials):
25       # Take username and password as input
26       # Look up the stored password from the credentials dictionary
27       # Compare stored password with entered password
28       return credentials.get(user) == password
29
30
31   # Google-style documentation format
32   def login_google(user, password, credentials):
33       """Validates user login using stored credentials.
34
35       Args:
36           user (str): Username provided for login.
37           password (str): Password entered by the user.
38           credentials (dict[str, str]): Dictionary storing valid credentials.
39
40       Returns:
41           bool: True if authentication is successful, False otherwise.
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   **TERMINAL**   PORTS   POSTMAN CONSOLE                    + ∨

```
making it easier for new developers to understand the function quickly.
PS C:\Users\Dell\Desktop\AI> []
```

```
ai(6.3 task -2).py      ai(6.3 task-3).py      ai 9.3(task-1)   ⠿ ‖ ↻ ↓ ↑ ↺ ☐   Launch ▾   <-3).py      ai 9.1(task-1).py      ai 9.1(ta

ai 9.1(task-2).py > ...
32    def login_google(user, password, credentials):
43        return credentials.get(user) == password
44
45
46    # Example data to test the functions
47    credentials_data = {
48        "admin": "1234",
49        "user1": "pass",
50        "john": "abcd"
51    }
52
53    print("Docstring Version:", login_docstring("admin", "1234", credentials_data))
54    print("Inline Version:", login_inline("user1", "wrong", credentials_data))
55    print("Google Version:", login_google("john", "abcd", credentials_data))
56
57
58    # Comparing the three documentation approaches
59
60    print("\nComparison of Documentation Styles:\n")
61
62    print("Docstring:")
63    print("Structured and visible using help().")
64    print("Good for reusable functions but may become lengthy.\n")
65
66    print("Inline Comments:")
67    print("Easy to understand and helpful for simple logic.")
68    print("Not structured and cannot be accessed using help().\n")
69
70    print("Google-Style:")
71    print("Professional and clearly separates inputs and outputs.")
72    print("Best suited for team projects and larger applications.\n")
73
74
75    # Recommendation for onboarding new developers
76
77    print("Recommended Style: Google-style documentation.")
78    print("Reason: It clearly explains parameters and return values,")
79    print("making it easier for new developers to understand the function quickly.")
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   POSTMAN CONSOLE

```
making it easier for new developers to understand the function quickly.
PS C:\Users\Dell\Desktop\AI>
```

**OUTPUT:**

```
- Best suited for Mathematical Utilities libraries
PS C:\Users\Dell\Desktop\AI>  c:; cd 'c:\Users\Dell\Desktop\AI'; & 'C:\Python314\python.exe' 'c:\Users\Dell\.vscode\extensions\ms-pytho
n.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56666' '--' 'c:\Users\Dell\Desktop\AI\ai 9.1(task-2).py'
Docstring Version: True
Inline Version: False
Google Version: True

Comparison of Documentation Styles:

Docstring:
Structured and visible using help().
Good for reusable functions but may become lengthy.

Inline Comments:
Good for reusable functions but may become lengthy.

Inline Comments:

Inline Comments:
Inline Comments:
Easy to understand and helpful for simple logic.
Not structured and cannot be accessed using help().


Google-Style:
Professional and clearly separates inputs and outputs.
Best suited for team projects and larger applications.

Recommended Style: Google-style documentation.
Recommended Style: Google-style documentation.
Reason: It clearly explains parameters and return values,
making it easier for new developers to understand the function quickly.
PS C:\Users\Dell\Desktop\AI> 
```

## EXPLANATION:

Docstrings are written inside triple quotes below a function. They explain the purpose, parameters, and return values clearly. They can be accessed using help() and are useful for documentation tools. This style is good for reusable and well-structured code.

### Inline Comment Style – Explanation

Inline comments use # to explain specific lines of code. They help understand the internal logic step-by-step. However, they are not structured and cannot be accessed as formal documentation. They are best for explaining simple or complex logic inside the function.

### Google-Style Documentation – Explanation

Google-style documentation is a structured docstring format with clear sections like Args and Returns. It improves clarity and consistency in team projects. It is widely used in industry and suitable for larger applications.

### Recommendation

Google-style documentation is best for new developers because it clearly explains inputs and outputs in a structured way, making the function easy to understand quickly

# Problem 3: Calculator (Automatic Documentation Generation)

Task: Design a Python module named calculator.py and

demonstrate automatic documentation generation.

Instructions:

1. Create a Python module calculator.py that includes the

following functions, each written with appropriate docstrings:

o add(a, b) – returns the sum of two numbers

o subtract(a, b) – returns the difference of two numbers

o multiply(a, b) – returns the product of two numbers

o divide(a, b) – returns the quotient of two numbers

2. Display the module documentation in the terminal using

Python's documentation tools.

3. Generate and export the module documentation in HTML

format using the pydoc utility, and open the generated HTML

file in a web browser to verify the output.

## PROMT:

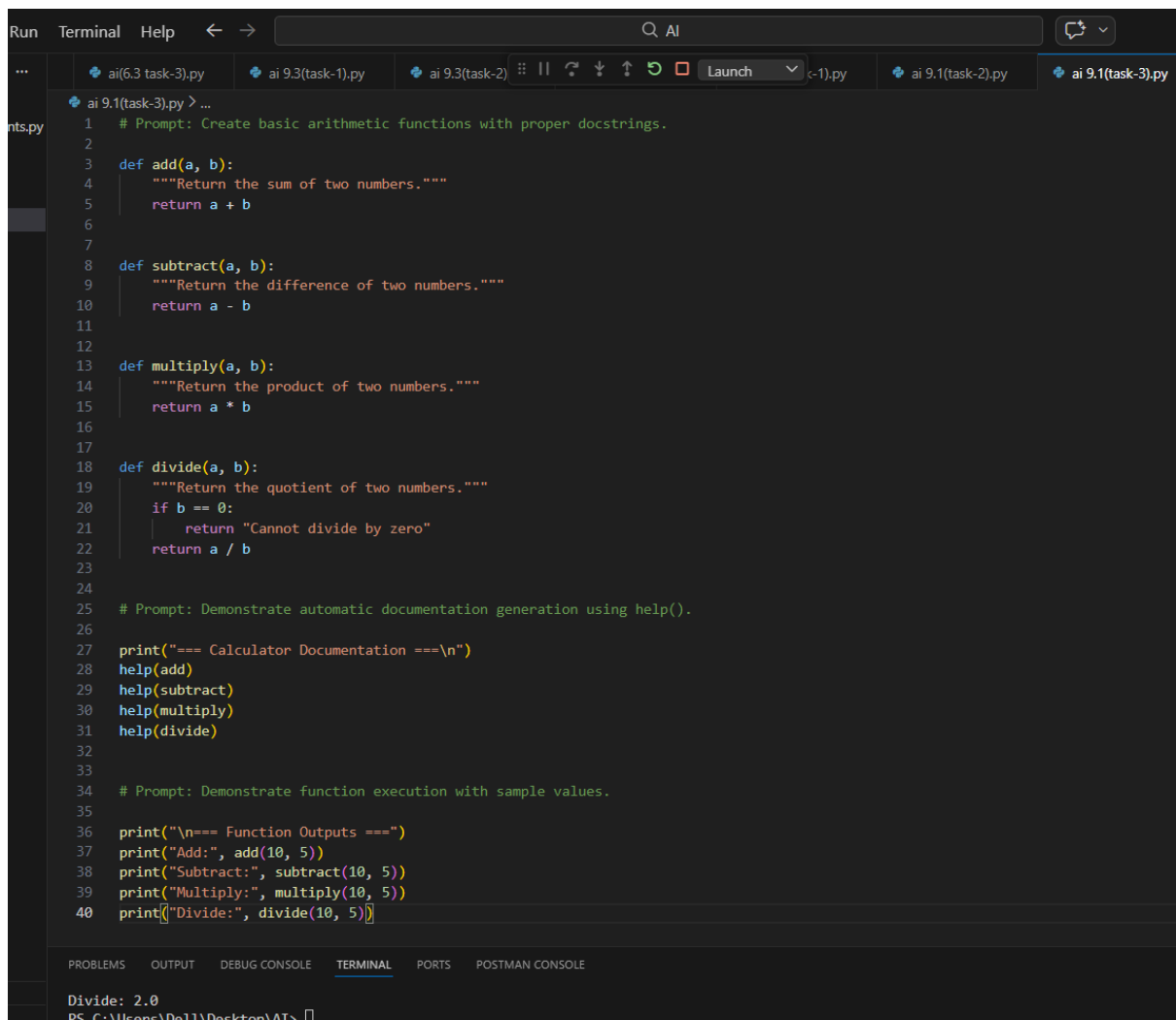 # Prompt: Create basic arithmetic functions (add, subtract, multiply, divide).

# Prompt: Add simple docstrings to each function for automatic documentation generation.

# Prompt: Use help() to display documentation of each function in the terminal.

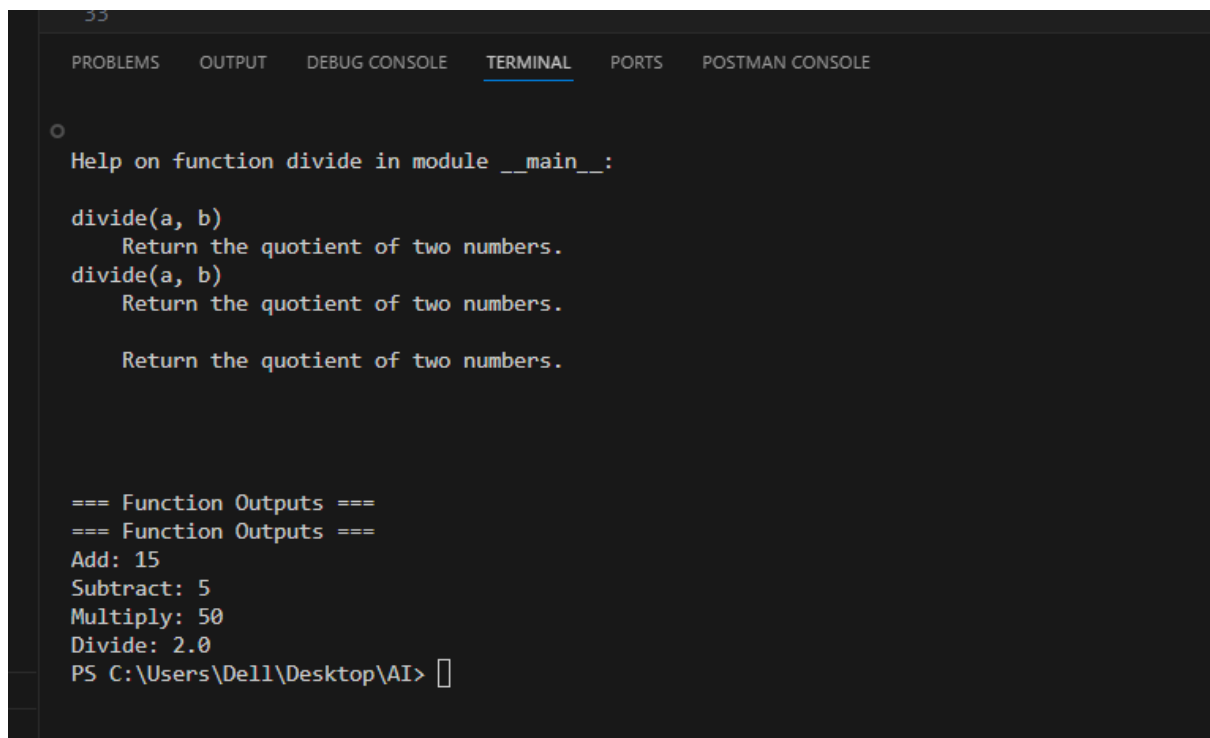# Prompt: Execute each function with sample input values.

# Prompt: Display the output of all arithmetic operations clearly.


## INPUT :

```
# Prompt: Create basic arithmetic functions with proper docstrings.

def add(a, b):
    """Return the sum of two numbers."""
    return a + b


def subtract(a, b):
    """Return the difference of two numbers."""
    return a - b


def multiply(a, b):
    """Return the product of two numbers."""
    return a * b


def divide(a, b):
    """Return the quotient of two numbers."""
    if b == 0:
        return "Cannot divide by zero"
    return a / b


# Prompt: Demonstrate automatic documentation generation using help().

print("=== Calculator Documentation ===\n")
help(add)
help(subtract)
help(multiply)
help(divide)


# Prompt: Demonstrate function execution with sample values.

print("\n=== Function Outputs ===")
print("Add:", add(10, 5))
print("Subtract:", subtract(10, 5))
print("Multiply:", multiply(10, 5))
print("Divide:", divide(10, 5))
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   POSTMAN CONSOLE

```
Divide: 2.0
PS C:\Users\Dell\Desktop\AI>
```

**OUTPUT:**

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   POSTMAN CONSOLE

```
Help on function divide in module __main__:

divide(a, b)
    Return the quotient of two numbers.
divide(a, b)
    Return the quotient of two numbers.

    Return the quotient of two numbers.




=== Function Outputs ===
=== Function Outputs ===
Add: 15
Subtract: 5
Multiply: 50
Divide: 2.0
PS C:\Users\Dell\Desktop\AI>
```

**EXPLANATION:**

**Docstring Explanation**

In this program, each arithmetic function contains a simple docstring that briefly explains its purpose. The docstring is written inside triple quotes directly below the function definition. When the help() function is called, Python automatically displays this documentation. This demonstrates how docstrings support automatic documentation generation.

**Importance of Using Docstrings**

Docstrings make functions self-explanatory and improve readability. They allow developers to understand what a function does without reading its implementation. Using help() is especially useful in larger projects, libraries, and team environments. This approach ensures clean, professional, and maintainable code

# Problem 4: Conversion Utilities Module

# Task:

1. Write a module named conversion.py with functions:

o decimal_to_binary(n)

o binary_to_decimal(b)

o decimal_to_hexadecimal(n)

2. Use Copilot for auto-generating docstrings.

3. Generate documentation in the terminal.

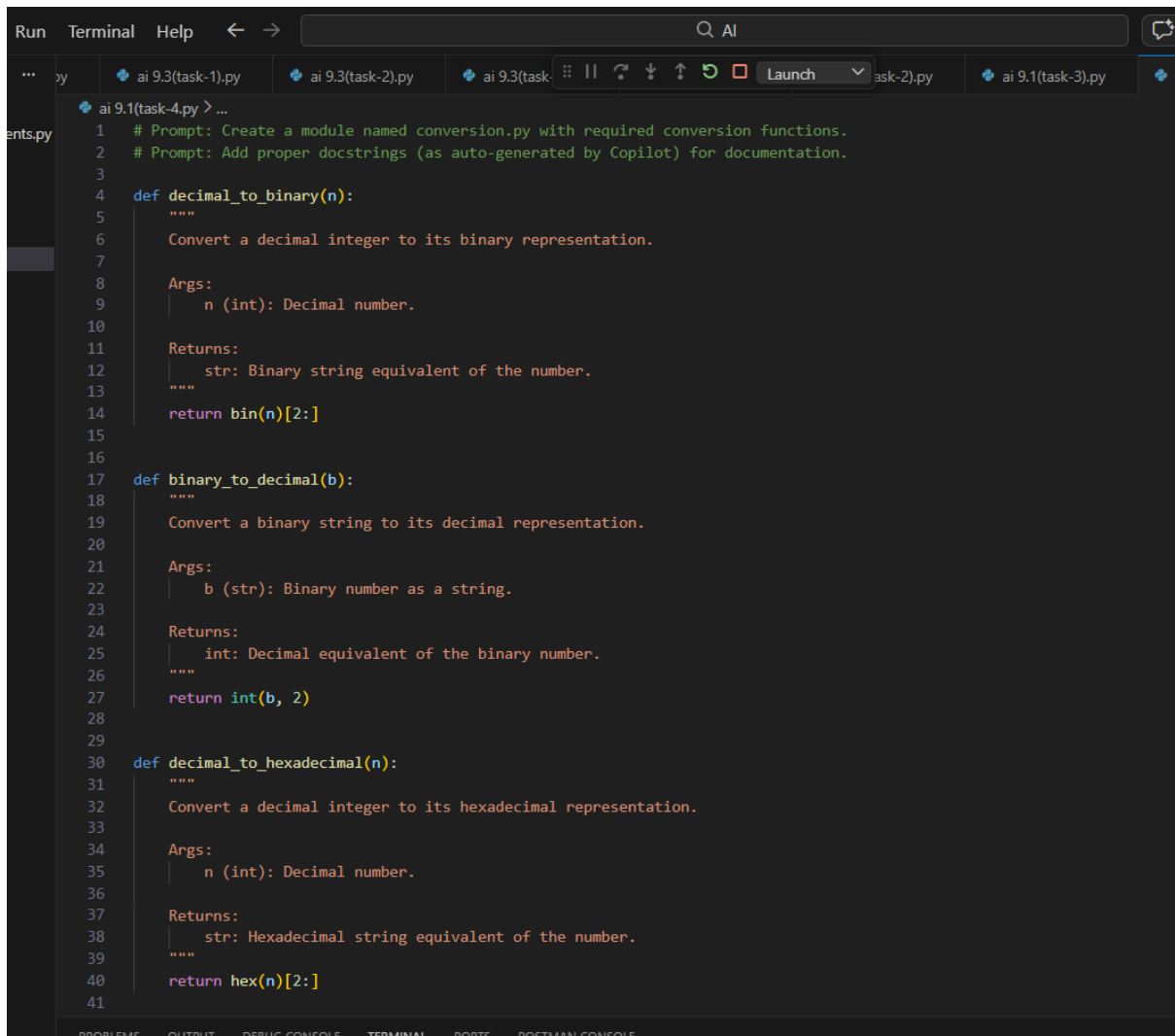4. Export the documentation in HTML format and open it in a

Browser

**PROMPTS:**

# Create a Python module named conversion.py with number conversion functions.

# Implement decimal_to_binary(n) to convert a decimal number to binary.

# Implement binary_to_decimal(b) to convert a binary string to decimal.

# Implement decimal_to_hexadecimal(n) to convert a decimal number to hexadecimal.

# Add proper docstrings to all functions (as if generated by Copilot).

# Use help() to display function documentation in the terminal.

# Demonstrate the conversion functions using sample inputs.

# Generate module documentation in HTML format using the pydoc utility.

# Open the generated HTML file in a web browser to verify the documentation.

## INPUT:

```
Run   Terminal   Help   ←   →                                    Q AI

...  py    ● ai 9.3(task-1).py   ● ai 9.3(task-2).py   ● ai 9.3(task-   :: II  ⟳  ↓  ↑  ↺  □  Launch  ∨  ask-2).py   ● ai 9.1(task-3).py   ●
         ● ai 9.1(task-4.py > ...
ents.py    1   # Prompt: Create a module named conversion.py with required conversion functions.
           2   # Prompt: Add proper docstrings (as auto-generated by Copilot) for documentation.
           3
           4   def decimal_to_binary(n):
           5       """
           6       Convert a decimal integer to its binary representation.
           7
           8       Args:
           9           n (int): Decimal number.
          10
          11       Returns:
          12           str: Binary string equivalent of the number.
          13       """
          14       return bin(n)[2:]
          15
          16
          17   def binary_to_decimal(b):
          18       """
          19       Convert a binary string to its decimal representation.
          20
          21       Args:
          22           b (str): Binary number as a string.
          23
          24       Returns:
          25           int: Decimal equivalent of the binary number.
          26       """
          27       return int(b, 2)
          28
          29
          30   def decimal_to_hexadecimal(n):
          31       """
          32       Convert a decimal integer to its hexadecimal representation.
          33
          34       Args:
          35           n (int): Decimal number.
          36
          37       Returns:
          38           str: Hexadecimal string equivalent of the number.
          39       """
          40       return hex(n)[2:]
          41

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POSTMAN CONSOLE
```

## OUTPUT:

```
 28

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POSTMAN CONSOLE

Decimal to Binary (10): 1010
Binary to Decimal (1010): 10
Binary to Decimal (1010): 10
Decimal to Hexadecimal (255): ff
PS C:\Users\Dell\Desktop\AI> |
```

## Explanation:

Each function in the module contains a structured docstring that explains its purpose, input arguments, and return values. These docstrings follow a clear format similar to professional documentation standards. When help() is used, Python automatically reads and displays these docstrings in the terminal. This makes the module self-documented and easy to understand.

Automatic Documentation Generation

The help() function demonstrates how Python can generate documentation directly from docstrings. Additionally, using the pydoc -w conversion command creates an HTML file containing the module documentation. This shows how properly written docstrings support both terminal-based and web-based documentation generation, making the code more professional and maintainable.

## Problem 5 – Course Management Module

## Task:

1. Create a module course.py with functions:

o add_course(course_id, name, credits)

o remove_course(course_id)

o get_course(course_id)

2. Add docstrings with Copilot.

3. Generate documentation in the terminal.

4. Export the documentation in HTML format and open it in a

browser.
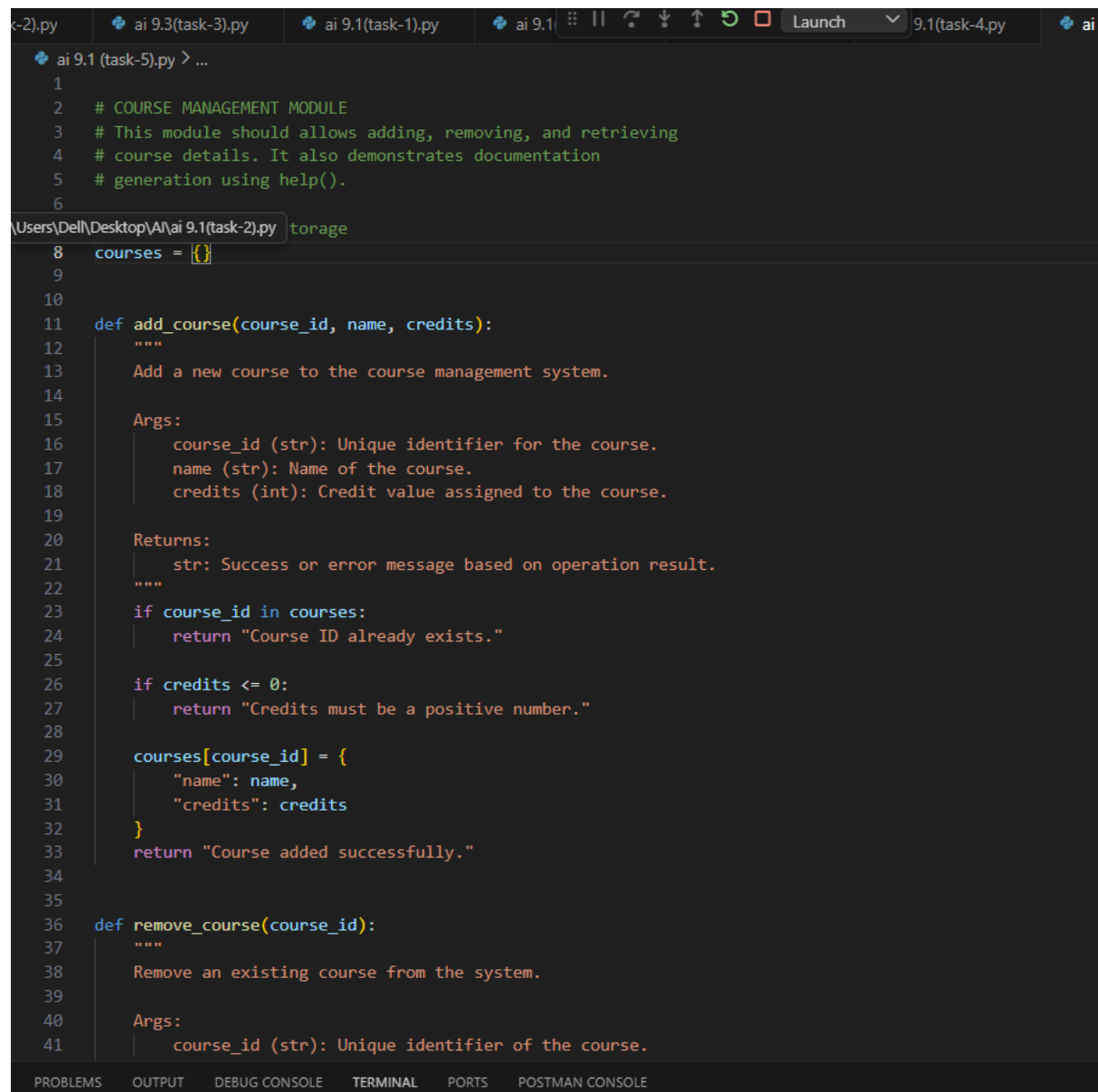
**PROMPTS:**

COURSE MANAGEMENT MODULE

# This module should allows adding, removing, and retrieving

# course details. It also demonstrates documentation

# generation using help().

# In-memory course storage

**INPUT:**

```python
# COURSE MANAGEMENT MODULE
# This module should allows adding, removing, and retrieving
# course details. It also demonstrates documentation
# generation using help().

# In-memory course storage
courses = {}


def add_course(course_id, name, credits):
    """
    Add a new course to the course management system.

    Args:
        course_id (str): Unique identifier for the course.
        name (str): Name of the course.
        credits (int): Credit value assigned to the course.

    Returns:
        str: Success or error message based on operation result.
    """
    if course_id in courses:
        return "Course ID already exists."

    if credits <= 0:
        return "Credits must be a positive number."

    courses[course_id] = {
        "name": name,
        "credits": credits
    }
    return "Course added successfully."


def remove_course(course_id):
    """
    Remove an existing course from the system.

    Args:
        course_id (str): Unique identifier of the course.
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   **TERMINAL**   PORTS   POSTMAN CONSOLE

ai 9.1 (task-5).py > ...

```python
ents.py
        36    def remove_course(course_id):
        43        Returns:
        44            str: Success or error message.
        45        """
        46        if course_id in courses:
        47            del courses[course_id]
        48            return "Course removed successfully."
        49        return "Course not found."
        50
        51
        52    def get_course(course_id):
        53        """
        54        Retrieve course details using course ID.
        55
        56        Args:
        57            course_id (str): Unique identifier of the course.
        58
        59        Returns:
        60            dict or str: Course details if found, otherwise error message.
        61        """
        62        return courses.get(course_id, "Course not found.")
        63
        64
        65
        66    if __name__ == "__main__":
        67
        68        print("=============================================")
        69        print("       COURSE MANAGEMENT SYSTEM")
        70        print("=============================================")
        71
        72        while True:
        73
        74            print("\nPlease choose an option below:")
        75            print("1. Add a New Course")
        76            print("2. Remove an Existing Course")
        77            print("3. View Course Details")
        78            print("4. Show Function Documentation")
        79            print("5. Exit")
        80
        81            choice = input("Enter your choice (1-5): ")
        82
```
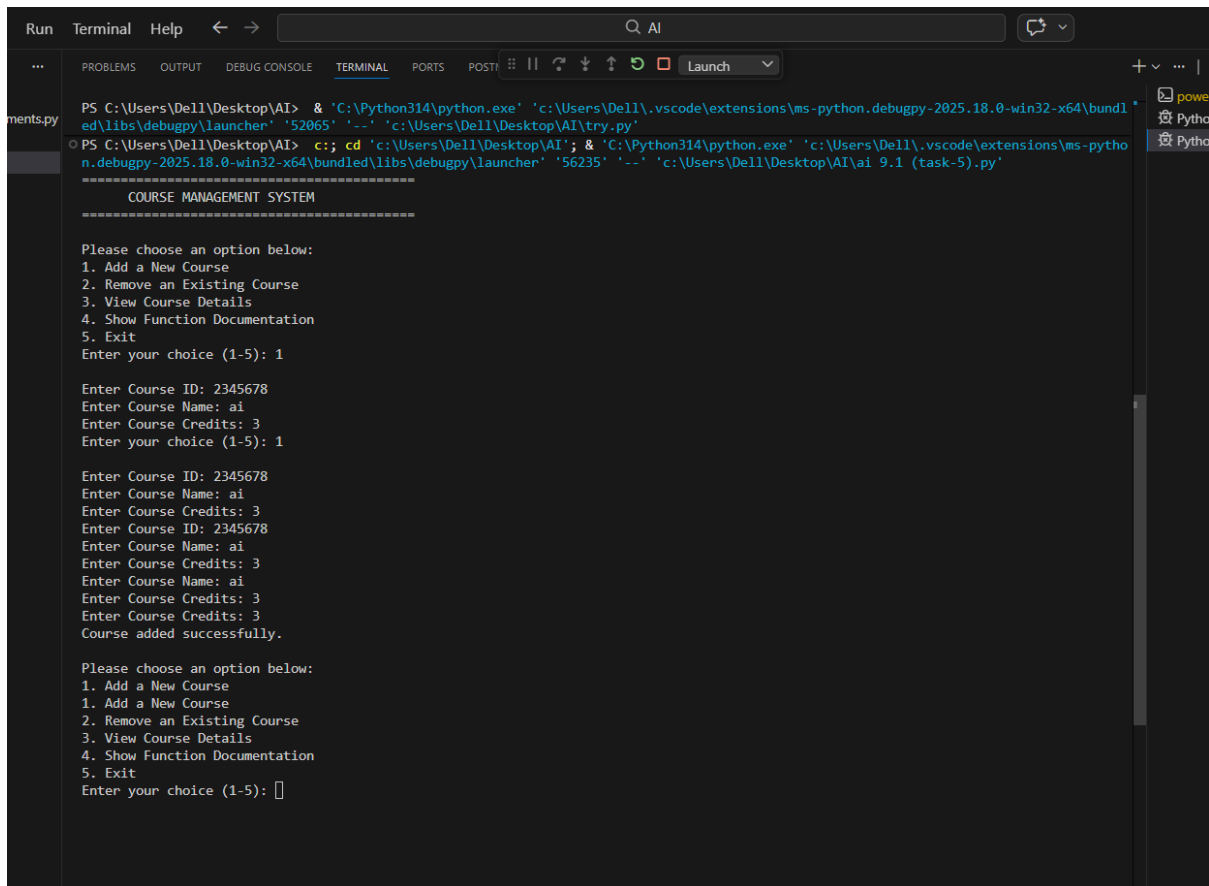
PROBLEMS    OUTPUT    DEBUG CONSOLE    **TERMINAL**    PORTS    POSTMAN CONSOLE
...
5. Exit

```python
109        # GET COURSE DETAILS
110
111    elif choice == "3":
112
113        course_id = input("\nEnter Course ID to view details: ").strip()
114        result = get_course(course_id)
115
116        if isinstance(result, dict):
117            print("\nCourse Details:")
118            print("Course ID :", course_id)
119            print("Name       :", result["name"])
120            print("Credits    :", result["credits"])
121        else:
122            print(result)
123
124
125        # SHOW DOCUMENTATION USING help()
126
127    elif choice == "4":
128
129        print("\n=== FUNCTION DOCUMENTATION ===\n")
130        help(add_course)
131        help(remove_course)
132        help(get_course)
133
134        print("\nTo export documentation in HTML format, run this in terminal:")
135        print("pydoc -w course")
136
137        print("\nThen open the generated file using:")
138        print("Windows : start course.html")
139        print("Mac     : open course.html")
140
141
142
143    elif choice == "5":
144        print("\nExiting Course Management System...")
145        break
146
147    else:
148        print("Invalid choice. Please select between 1 and 5.")
```

**OUTPUT:**

## Explanation:

This program is a simple course management system that allows users to add, remove, and view course details through a menu-driven interface. It stores course information in a dictionary and performs basic validation like checking duplicate IDs and valid credits. Each function includes proper docstrings to support documentation generation. The documentation can be viewed using help() or exported to HTML format using the pydoc command.